

TSEA28 Datorteknik Y, lösningar till tentamen 18-10-31

1. a) Kopiera GRx till AR, räkna upp, flytta resultat till GRx, hoppa till start om inte noll, annars flytta PC till AR, lägg till A-fältet och uppdatera PC. Väljer placering av koden på mikrokodsadress 16 (allt från 10 upp till 122 skulle fungera).

Adress	Mikrokod	Kommentar
16	buss:=GRx, AR:=buss, R:=0, S:=0, uPC:=uPC+1	; Läs GRx till AR ;
17	buss:=uM(värdet 1), AR:=AR+buss, uPC:=uPC+1	; Räkna upp AR, påverka flaggor ;
18	buss:=AR, GRx:=buss, R:=0, S:=1, uPC:=0 om Z=1 annars uPC:=uPC+1	; Spara uppräknade värdet i GRx ; Nästa instruktion om resultat = 0
19	buss:=PC, AR:=buss, uPC:=uPC+1	; Hämta PC för att addera A-fältet
20	buss:=IR, AR:=AR+buss, uPC:=uPC+1	; Räkna vart hopp ska ske
21	buss:=AR, PC:=buss, uPC:=0	; Uppdatera programräknare, ta nästa ; instruktion

b) Adresseringsmod väljs från M-bitarna som mha K2 hoppar till rätt rutin

Adress	Värde
00	0000011
01	0000100
10	0000101
11	0000111

2. a) En VLIW-processor utför långa instruktionsord uppbyggda av ett flertal små instruktioner i varje klockcykel. Dessa långa instruktionsord sätts samman vid kompileringen. En superskalär processor utför flera små instruktioner i varje klockcykel, men det är processorn själv som väljer vilka instruktionerna ska vara.

b) SRAM = Statiskt RAM, DRAM = Dynamiskt RAM. Dynamiskt RAM behöver uppräshas (refresh) av alla bitar i minnet mha läsningar av data för att behålla informationen. Ett SRAM behöver inte detta. Ett SRAM är snabbare än ett DRAM, men drar också mer energi per bit, och tar mer yta på chipet per bit.

c) Ett virtuellt minne kräver en MMU (Memory Management Unit) som översätter virtuell adress till fysisk adress. Ett program kan med hjälp av MMU hindras att komma åt det fysiska minne som används av ett annat program, och därmed ökar säkerheten.

d) Eftersom en olika processorer tar olika antal klockcykler för att utföra en operation så kan en processor med låg klockfrekvens hinna med mer beräkningar än en processor med hög klockfrekvens, eftersom processorn med hög klockfrekvens kanske behöver många fler klockcykler för att genomföra en beräkning.

e) Antal pipelinesteg kommer inte öka klockfrekvensen motsvarande del (fördröjningen mellan varje pipelinesteg är olika, och klockfrekvensen begränsas av den längsta fördröjningen). Dessutom ger pipelining problem med olika typer av konflikter (styr, data etc.) så processorn måste i många fall vänta på att föregående instruktion ska slutföras. Detta ger bubblor i pipelining som motsvarar en sänkt prestanda.

3. Notering om tentauppgiften: data i resultatet på adress 0x2000101a och framåt är fel. Datat skulle vara 0x88 0x88 0x55 0x55 0x66 0x66.

TSEA28 Datorteknik Y, lösningar till tentamen 18-10-31

Subrutinen är enklast att beskriva som en loop över de olika byte:n i nyckeln, och sedan räkna upp indata och utdataposition, samt räkna ned antal tecken som ska kodas.

ARM Cortex-M4: Två lösningar där 1:a är mer skrivning men enklare att förstå, 2:a är mer kompakt.

```

Subrutin:  ldrb  r4,[r0],#1    ; r4 = 1:a byte av nyckel
           ldrb  r5,[r0],#1    ; r5 = 2:a byte av nyckel
           ldrb  r6,[r0],#1    ; r6 = 3:e byte av nyckel
           ldrb  r7,[r0]      ; r7 = 4:e byte av nyckel
loop:     ldrb  r8,[r1],#1    ; hämta ett tecken
           xor   r8,r8,r4      ; r8 = r8 xor 1:a byte i kod
           strb  r8,[r3],#1    ; Spara kodat tecken i minne
           subs r2,r2,#1      ; Räkna ned antal tecken kvar
           beq  done          ; hoppa ur om klar
           ldrb  r8,[r1],#1    ; hämta tecken
           xor   r8,r8,r5      ; r8 = r8 xor 2:a byte i kod
           strb  r8,[r3],#1    ; spara kodat tecken i minne
           subs r2,r2,#1      ; räkna ned antal tecken kvar
           beq  done          ; hoppa ur om klar
           ldrb  r8,[r1],#1    ; hämta tecken
           xor   r8,r8,r6      ; r8 = r8 xor 3:e byte i kod
           strb  r8,[r3],#1    ; spara kodat tecken i minne
           subs r2,r2,#1      ; räkna ned antal tecken kvar
           beq  done          ; hoppa ur om klar
           ldrb  r8,[r1],#1    ; hämta tecken
           xor   r8,r8,r7      ; r8 = r8 xor 4:e byte i kod
           strb  r8,[r3],#1    ; spara kodat tecken i minne
           subs r2,r2,#1      ; räkna ned antal tecken kvar
           bne  loop          ; nästa varv i kod om finns tecken kvar
done:     bx   lr             ; klar med alla tecken

Subrutin:  ldrb  r4,#0        ; Offset i nyckel
nextkey:  ldrb  r5,[r1],#1    ; hämta ett tecken
           xor   r5,[r0,r4]    ; hämta rätt byte i nyckel, bitvis xor
           strb  r5,[r3],#1    ; spara kodat tecken i minnet
           subs r2,r2,#1      ; minska antal tecken kvar
           beq  done          ; Hoppa ur subrutin om klar
           add  r4,r4,#1      ; Öka position i nyckel
           cmp  r4,#4         ; Hamnat utanför?
           beq  Subrutin      ; Börja med 1:a byte i nyckel igen
           b   nextkey        ; Ta nästa tecken att koda
done:     bx   lr             ; klar med subrutinen

```

68000:

```

Subrutin:  move.l #0,D1      ; Position i nyckeln
loop:     move.b D2,(A1)+    ; hämta tecken, peka på nästa
           xor.b (A0,D1),D2  ; xor mellan nyckelbyte och tecken
           move.b D2,(A2)+    ; spara kodat tecken i minnet
           sub.l #1,D0        ; Kontrollera om fler tecken att koda

```

TSEA28 Datorteknik Y, lösningar till tentamen 18-10-31

```

                beq    done        ; klar, hoppa ur
                add.l  #1,D1       ; Öka position i nyckel
                cmp.l  #4,D1       ; Hamnat utanför nyckel?
                beq    Subrutin    ; ja återställ till början av nyckel, nästa tecken
                bra    loop        ; nej, ta nästa tecken
done:          rts                ; klar med subrutinen
    
```

4. Avbrottsrutin så alla generella register måste återställas efter anrop. För ARM Cortex-M görs detta automatiskt för R0-R3,R12 och LR.

Arm Cortex-M4:

```

inputport .field 0x40001000,32
dataaddr  .field 0x40001010,32

avbrott: ldr    r1,inputport ; peka på indataportens adress
loop     ldrb   r0,[r1]      ; hämta en byte från 0x40001000
         and    r0,r0,#3     ; Plocka fram bit 0 och 1 ur inläst värde
         cmp    r0,#3       ; kontrollera att båda är 1
         bne   loop        ; om inte läs dom igen
         ldr   r1,dataaddr ; peka på nästa port att läsa ifrån
         ldr   r0,[r1]     ; Hämta data från port
         push  {r4,r5,r6,r7,lr} ; Spara register som förstörs av subrutin
         bl    Subrutin2   ; anropa subrutin
         push  {r4,r5,r6,r7,lr} ; Återställ register
         bx   lr           ; avsluta avbrottet
    
```

68000:

```

Avbrott: move.l D0,-(A7)      ; push D0 på stack
         move.l D1,-(A7)      ; push D1 på stack
         move.l D2,-(A7)      ; push D2 på stack
         move.l D3,-(A7)      ; push D3 på stack
         move.l D4,-(A7)      ; push D4 på stack
         move.l D5,-(A7)      ; push D5 på stack
         move.l D6,-(A7)      ; push D6 på stack
         move.l D7,-(A7)      ; push D7 på stack
loop:    move.b $40001000,D0   ; Hämta värde
         and.b  #3,D0         ; Plocka fram bit 0 och 1
         cmp    #3,D0         ; Kontrollera att båda bitar är 1
         bne   loop          ; om inte läsa en gång till
         move.l $40001010,D0   ; läs indata till subrutin
         jsr   Subrutin2
         move.l (A7)+,D7       ; återställ registren
         move.l (A7)+,D6
         move.l (A7)+,D5
         move.l (A7)+,D4
         move.l (A7)+,D3
         move.l (A7)+,D2
         move.l (A7)+,D1
    
```

```

move.l (A7)+,D0
rte ; klar med avbrottet
    
```

5. a) $0x2e7 = 0010\ 1110\ 0111 = 010\ 1110\ 0111$ (ska vara 11 bitar, inte 12).

b) $0x6223 = 0110\ 0010\ 0010\ 0011$, $0xbdac = 1011\ 1101\ 1010\ 1100$

```

Carry: 111      1
        0110001000100011
+       1011110110101100
-----
        0001111111001111
    
```

$Z = 0, C = 1, N = 0, O = 0.$

Motivering: Eftersom svar inte är 0 $\Rightarrow Z=0$, Carry ut $\Rightarrow C=1$, teckenbit (MSB) = 0 $\Rightarrow N=0$, och om indata tolkats som 2-komplementstal skulle ett positivt och negativt tal adderas \Rightarrow belopp hos slutresultat måste vara mindre än indatatermernas belopp \Rightarrow inget spill $\Rightarrow O=0$. Alternativt: carry in till MSB är 1 och carry ut från MSB är 1 $\Rightarrow O = 0$.

d) Addition av 2-komplementstal \Rightarrow måste teckenförlänga innan addition

```

Carry: 11111111111111
        1111111110110101
+       0011010111011110
-----
        0011010110010011
    
```

6. a) 7 bitars index \Rightarrow 128 cachelines i varje väg. Varje cacheline är 64 byte \Rightarrow varje väg lagrar totalt $64 \cdot 128 = 8192$ byte. Antal vägar = totalt minne/minne per väg = $524288/8192 = 64$.

Notera att detta inte är korrekt för en Raspberry PI 3. I Raspberry PI 3 används en ARM A53 processor, som har en 16-vägs Level 2 cache (dvs 9 bitar används för index i den).

b) Enligt uppgiften 7 bitar index, cachelinelängd 64 motsvarar 6 bitar för att peka ut byte i cacheline. $32-7-6=19$ bitar tag.

c) Binär beskrivning av första adress och andra adress i läsningen av indata, med markering av gränser mellan tag (19 bit), index (7 bit) och byteposition (6 bit):

```

0010 0000 0000 0000 000|0 0000 00|00 0000
0010 0000 0000 0000 000|0 0010 00|00 0000
    
```

Eftersom uppräknings av adresser blir maximalt 1111000 i index-delen så är det 16 olika värden innan samma index fås igen. Det är 64 vägar i cachén $\Rightarrow 16 \cdot 64 = 1024$ läsningar innan data behöver kastas ut ur cache.